0.1 Nature de Git

Git est un gestionnaire de versions distribués (DVCS en anglais pour Distributed Version Control Systems) qui repose sur deux fonctionnalités majeures :

— Avec Git, les clients n'extraient plus seulement la dernière version d'un fichier, mais ils dupliquent compl'etement le dépôt. Ainsi, si le serveur disparaît et si les systèmes collaboraient via ce serveur, n'importe quel dépôt d'un des clients peut être copié sur le serveur pour le restaurer. Chaque extraction devient une sauvegarde complète de toutes les données

— Git considère les données. Les systèmes (tel que CVS, Subversion, Perforce, Bazaar et autres) considèrent l'information qu'ils gèrent comme une liste de fichiers et les modifications effectuées sur chaque fichier dans le temps. Git ne gère pas et ne stocke pas les informations de cette manière. À la place, il pense ses données plus comme un instantané d'un mini système de fichiers. À chaque fois que vous validez ou enregistrez l' état du projet dans Git, il prend effectivement un instantané du contenu de votre espace de travail à ce moment et enregistre une référence à cet instantané. Pour être efficace, si les fichiers n'ont pas changé, Git ne stocke pas le fichier à nouveau, juste une référence vers le fichier original qui n'a pas été modifié.

De plus avec Git :

— Presque toutes les opérations sont locales : Par exemple, pour parcourir l'historique d'un projet, Git n'a pas besoin d'aller le chercher sur un serveur pour vous l'afficher ; il n'a qu'à simplement le lire directement dans votre base de données locale. Ainsi on peut entièrement travailler hors-connexion

— Git gère l'intégrité Dans Git, tout est vérifié par une somme de contrôle avant d'être stocké. Par la suite cette somme de contrôle, signature unique, sert de référence. Cela signifie qu'il est impossible de modifier le contenu d'un fichier ou d'un répertoire sans que Git ne s'en aperçoive. Cette fonctionnalité est ancrée dans les fondations de Git et fait partie intégrante de sa philosophie. Vous ne pouvez pas perdre des données en cours de transfert ou corrompre un fichier sans que Git ne puisse le détecter

— Généralement, Git ne fait qu'ajouter des données Dès que vous avez validé un instantané dans Git, il est très difficile de le perdre

0.2 Les 3 état de Git

Ici, il faut être attentif. Il est primordial de se souvenir de ce qui suit si vous souhaitez que le reste de votre apprentissage s'effectue sans difficulté.

Git gère trois états dans lesquels les fichiers peuvent résider :

— Validé : signifie que les données sont stockées en sécurité dans votre base de données locale.

— Modifié : signifie que vous avez modifié le fichier mais qu'il n'a pas encore été validé en base.

— Indexé : signifie que vous avez marqué un fichier modifié dans sa version actuelle pour qu'il fasse partie du prochain instantané du projet.

Ceci nous mène aux trois sections principales d'un projet Git : le répertoire Git, le répertoire de travail et la zone d'index.

0.3 Configuration

Notre identité d'utilisateur :

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

L'éditeur de texte qui sera utilisé quand Git vous demande de saisir un message.

```
$ git config --global core.editor vim
```

Une autre option utile est le paramétrage de l'outil de différences à utiliser pour la résolution des conflits de fusion.

```
$ git config --global merge.tool vimdiff
```

0.4 Les branches

Les branches sont très utiles pour travailler sur des stories différente. L'idée est de créer une branche par story et d'y effectuer tout le travail relatif à celle-ci sans impacter directement la branche master.
Pour créer une branche on peut procéder comme suit :

```
$ git branch my_branch
$ git checkout my_branch
```

ou directement :

```
$ git checkout -b my_branch
Switched to a new branch my_branch
```

A partir de cet instant notre HEAD pointe sur cette nouvelle branche. Toutes les modifications qui y seront réalisées le seront uniquement dans celle-ci

On peut ainsi créer plusieurs branches en parallèle pour travailler sur différents sujets.

Quand le travail sur une branche est terminé, il suffit de commiter les modifications puis de merger la branche avec la branche master pour pouvoir les publier.
On commite

```
$ git commit -a -m 'commit message'
```

On rebascule sur la branche master

```
$ git checkout master
```

Et on merge

```
$ git merge my_branch
```

Il ne reste plus qu'à publier les modifications

```
$ git pull
```